Linux Kernel Training

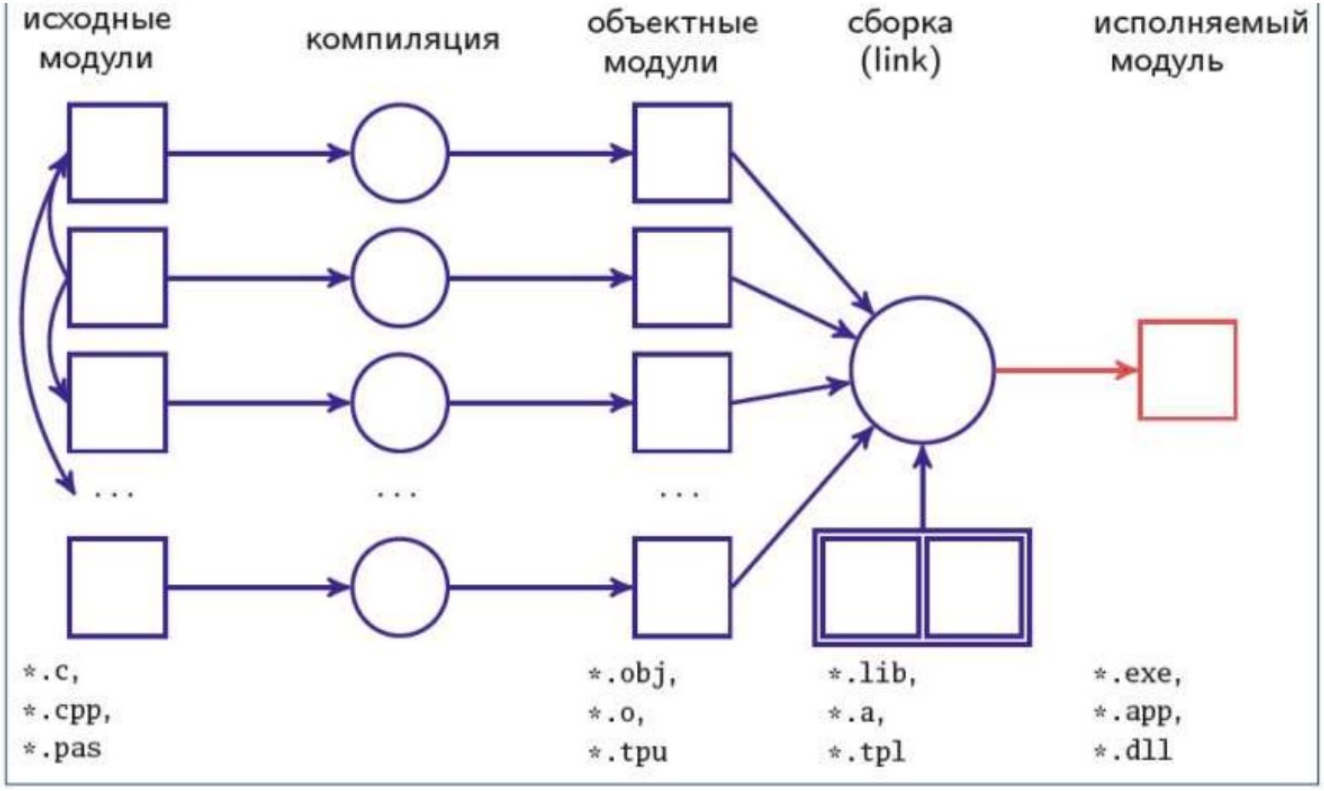
Make

Kharkiv, 2019

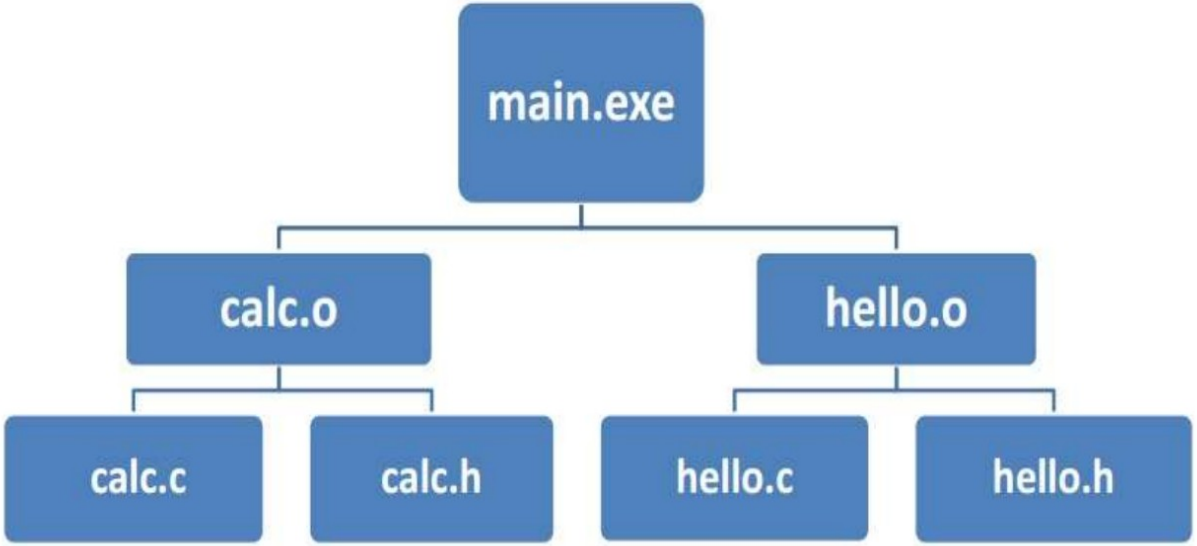
Agenda

1. Make Overview
2. Running of makefile
3. Make rules
4. Variables
5. Targets
6. Conditionals
7. Example

Сборка проекта



Пример модульной структуры



Сборка

Самый простой путь: перечислить компилятору все исходные файлы:

```
gcc ./calc.c ./hello.c ./main.c -o main.exe
```

Работает, но не самое лучшее решение. Почему?

Если в проекте тысячи строк кода и много исходных файлов, то каждая перекомпиляция будет длиться очень долго.

Сборка проекта

```
gcc -c ./calc.c  ./calc.o
gcc -c ./hello.c ./hello.o
gcc -c ./main.c  ./main.o
gcc ./calc.o ./hello.o ./main.o -o main.exe
```

Если теперь мы изменим, например, `calc.c`, то достаточно будет пересобрать соответствующий объектный файл и объединить все объектные файлы в исполняемый.

Всего два запуска компилятора
(вместо, скажем, тысячи ☺)

Было бы хорошо, если бы это делалось автоматически…

Make

GNU make — утилита, автоматизирующая процесс преобразования файлов из одной формы в другую.

- В нашем случае это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы.
- Производит эффективный обход дерева сборки, обновляя только ту часть, которая зависит от измененных файлов

Пример Makefile

```
main.exe: main.o calc.o hello.o
gcc ./main.o ./calc.o ./hello.o -o ./main.exe
calc.o: calc.c calc.h
gcc -Wall -c ./calc.c -o ./calc.o
hello.o: hello.c hello.h
gcc -Wall -c ./hello.c -o ./hello.o
main.o: main.c calc.h hello.h
gcc -Wall -c ./main.c -o ./main.o
```

При запуске make без параметров начнется сборка первой цели в файле, т.е. main.exe

По умолчанию используется имя файла Makefile. С помощью опции -f можно явно указать другой файл: make -f Makefile_simple

Run make and Naming conventions

GNUMakefile

makefile

Makefile

make -f anyname

Make rules

Makefile example:

prog : module1.o main.o

    gcc -o prog module1.o main.o

module1.o : module1.c module1.h

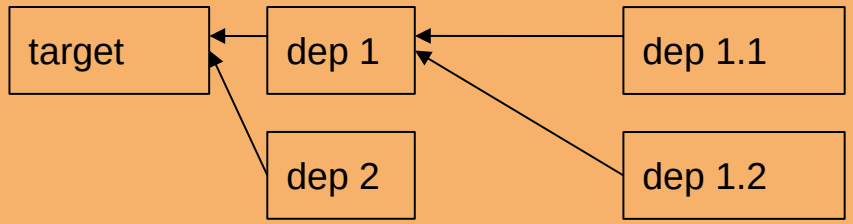
    gcc -c module1.c

main.o : main.c module1.h

    gcc -c main.c

target : dependencies

TAB commands #shell commands



Using variables

```
prog : module1.o main.o
    gcc -o prog module1.o main.o
module1.o : module1.c module1.h
    gcc -c module1.c
main.o : main.c module1.h
    gcc -c main.c
```

```
C=gcc
OBJS = module1.o main.o
prog : ${OBJS}
    ${C} -o prog ${OBJS}
module1.o : module1.c module1.h
    ${C} -c module1.c
main.o : main.c module1.h
    ${C} -c main.c
```

Running with variables

make C=gcc

C=gcc make

export C=gcc

make

Implicit rules

```
%.o : %.c
    $(CC) -c -g $<
```

Empty rule can protect your target file:

```
target: ;
```

Automatic variables

hello.o: hello.c hello.h

gcc -c hello.c

hello.o: hello.c hello.h

gcc -c $<

$@ - The name of the target of the rule (hello.o).

$< - The name of the first dependency (hello.c).

$^ - The names of all the dependencies (hello.c hello.h).

$? - The names of all dependencies that are newer than the target

Make options

-f filename - when the makefile name is not standard

-t - (touch) mark the targets as up to date

-q - (question) are the targets up to date, exits with 0 if true

-n - print the commands to execute but do not execute them

/ -t, -q, and -n, cannot be used together /

-s - silent mode

-k - keep going – compile all the prerequisites even if not able to link them

Phony targets

Target without dependencies

```
.PHONY : clean

clean:

    rm $(OBJS)
```

Standard targets

all *- Compile the entire program. Can be default target*

install *- Compile the program and copy the executables, libraries, and so on*

uninstall *- Delete all the installed files*

info *- Generate any Info files needed.*

check *- Perform self-tests (if any)*

clean *- Delete all files from the current directory*

dist *- Create a distribution tar file for this program*

distclean *- Delete all files from the current directory except distribution*

Contitionals

if ifeq ifneq ifdef ifndef
elif
else
endif

Example:

```
flags_for_gcc = -lgnu

flags_for_other =

ifeq ($(CC),gcc)

    flags=$(flags_for_gcc)

else

    flags=$(flags_for_other)

endif
```

Multithreading

make -jN

N - number of threads

make -j4

less time for compile, more time for finding the problem

Control make

$(error text...)

Generates a fatal error where the message is text.

$(warning text...)

This function works similarly to the error function, above, but program continue to work

Shell invoke
*files = $(shell echo *.c)*

Example

```
NAME = mr. Boss

SIGNER = mr. Signer

$(eval export DATE=$$(sh -c "LC_ALL=en_EN.utf8
date"))

# 1st rule is executed by default

letter_signed.txt: letter.txt

        cp $< $@

        echo "Signed by ${SIGNER} ${DATE}" >> $@

header_template.txt:

        echo "Dear %%NAME%%," >> $@

header.txt: header_template.txt

        sed "-es/%%NAME%%/${NAME}/" <$< >$@
```

```
body.txt:

        echo "Here is the letter provided for the testing purposes" > $@

footer.txt:

        echo " best regards," > $@

        echo " mr. Sender" >> $@

letter.txt: header.txt body.txt footer.txt

        cat $^ >$@

clean:

        rm *.txt
```

Bye